Homework 4

Author: Rudy Martinez

Date: 11/28/2021

Google Colab Link: https://colab.research.google.com/drive/1YhbIe5vd8KNGMcW72KFhy8_TdjmQsOxQ?usp=sharing

```
#Data Manipulation
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

#Machine Learning
import tensorflow as tf
from tensorflow import keras
from keras import layers, models
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import activations
from keras.layers import Activation, Dense
from sklearn.model_selection import train_test_split

#Datasets
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.datasets import cifar10
```

Part 1

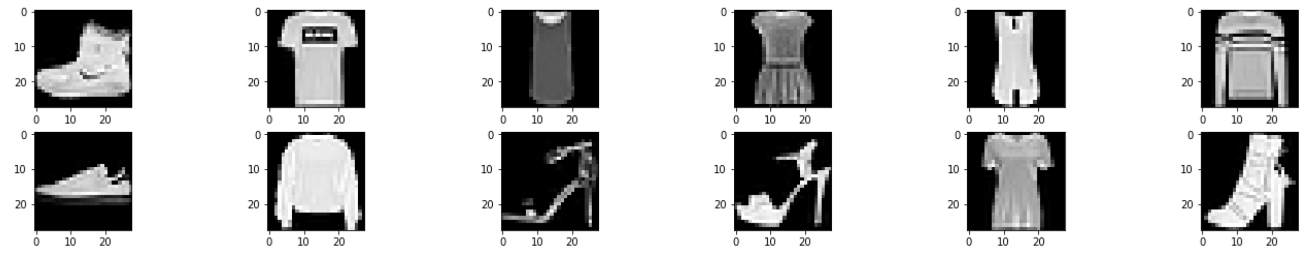
(5pt) Load the dataset using `tf.keras.datasets.fashion_mnist.load_data()` and show the first 12 images of the training dataset in two rows.

```
#Load Data
(X_train, y_train), (X_test, y_test)  = fashion_mnist.load_data()

#Train and Test Shape
print('Train: X=%s, y=%s' % (X_train.shape, y_train.shape))
print('Test: X=%s, y=%s' % (X_test.shape, y_test.shape))
```

```
    Train: X=(60000, 28, 28), y=(60000,)
    Test: X=(10000, 28, 28), y=(10000,)
```

```
#Display Training Dataset Images
```

```
plt.figure(figsize=(24, 4))

for i in range(12):
    plt.subplot(2, 12, 2*i+1)
    plt.imshow(X_train[i], cmap=plt.get_cmap('gray'))

plt.show()
```



(5pt) Add the "depth" dimension to the training/testing image data using .reshape(), use to_categorical() to transform all labels into their one-hot encoding forms, and normalize the pixel values of all images into [0, 1]. Print out the shapes of training and testing images.

- Note that the imported training/testing image data have a shape of (number_samples, image_height, image_width) and you want to reshape it into the shape of (number_samples, image_height, image_width, image_depth/image_channels)

```
#Add "depth" dimension to the training/testing image data
X_train = X_train.reshape((60000, 28, 28, 1))
X_test = X_test.reshape((10000, 28, 28, 1))

#To Categorical to transform all labels into one-hot encoded forms
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

#Normalize the pixel values of grayscale images, e.g. rescale them to the range [0,1]
#Convert the data type from unsigned integers to floats
#Divide the pixel values by the maximum value
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

#Print out shapes of training and testing images
print('Train: X=%s, y=%s' % (X_train.shape, y_train.shape))
print('Test: X=%s, y=%s' % (X_test.shape, y_test.shape))
```

```
    Train: X=(60000, 28, 28, 1), y=(60000, 10)
    Test: X=(10000, 28, 28, 1), y=(10000, 10)
```

(10pt) Build a CNN model using a stack of Conv2D (128 filters of size (3, 3) with ReLU activation), MaxPooling2D (pool size of (2, 2)), Conv2D (64 filters of size (3, 3 with ReLU activation), MaxPooling2D (pool size of (2, 2)), Dense (128 hidden units with ReLU activation), and output layer. Display the model architecture using .summary().

- You need to specify other parameters of the input layer and output layer.

```
#CNN Model Function
def model_defined():
  model = models.Sequential()

  #Conv2D (128 filters of size (3, 3) with ReLU activation)
  model.add(layers.Conv2D(128, (3, 3), activation='relu',input_shape=(28, 28, 1)))

  #MaxPooling2D (pool size of (2, 2))
  model.add(layers.MaxPooling2D((2, 2)))

  #Conv2D (64 filters of size (3, 3 with ReLU activation)
  model.add(layers.Conv2D(64, (3, 3), activation='relu'))

  #MaxPooling2D (pool size of (2, 2))
  model.add(layers.MaxPooling2D((2, 2)))

  model.add(layers.Flatten())

  #Dense (128 hidden units with ReLU activation)
  model.add(layers.Dense(128, activation='relu'))

  model.add(layers.Dense(10, activation='sigmoid'))

  return model
```

```
#Execute Function
cnn_model = model_defined()
cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 128)       1280

 max_pooling2d (MaxPooling2D  (None, 13, 13, 128)      0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        73792

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0
 dense (Dense)               (None, 128)               204928

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 281,290
Trainable params: 281,290
Non-trainable params: 0
_________________________________________________________________
```

(10pt) Compile and train the model for 10 epochs and batch size of 32. Set verbose = 0 during the training to compress the training progress.

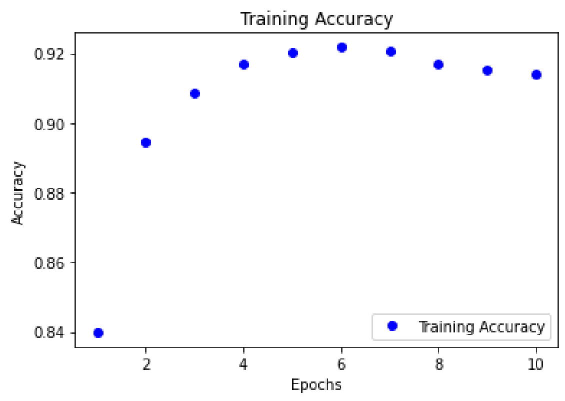
Draw the plot of the training accuracy w.r.t. the epoch number

```
#Compile Model
cnn_model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

#Train Model with parameters
history = cnn_model.fit(X_train, y_train,
                    epochs=10,
                    batch_size=32)
```

```
Epoch 1/10
1875/1875 [==============================] - 171s 91ms/step - loss: 0.0799 - accuracy: 0.8399
Epoch 2/10
1875/1875 [==============================] - 166s 89ms/step - loss: 0.0524 - accuracy: 0.8944
Epoch 3/10
1875/1875 [==============================] - 165s 88ms/step - loss: 0.0461 - accuracy: 0.9088
Epoch 4/10
1875/1875 [==============================] - 163s 87ms/step - loss: 0.0427 - accuracy: 0.9168
Epoch 5/10
1875/1875 [==============================] - 163s 87ms/step - loss: 0.0416 - accuracy: 0.9202
Epoch 6/10
1875/1875 [==============================] - 164s 88ms/step - loss: 0.0415 - accuracy: 0.9218
Epoch 7/10
1875/1875 [==============================] - 163s 87ms/step - loss: 0.0433 - accuracy: 0.9205
Epoch 8/10
1875/1875 [==============================] - 162s 86ms/step - loss: 0.0453 - accuracy: 0.9168
Epoch 9/10
1875/1875 [==============================] - 162s 86ms/step - loss: 0.0465 - accuracy: 0.9152
Epoch 10/10
1875/1875 [==============================] - 160s 86ms/step - loss: 0.0473 - accuracy: 0.9141
```

```
#Create dictionary for loss and accuracy keys (for charting)
history_dict = history.history

#Plotting
plt.clf()
acc = history_dict['accuracy']
loss = history.history['loss']
```

```
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



(5pt) Test your trained model on the testing dataset and observe the loss and accuracy using .evaluate().

```
loss, accuracy = cnn_model.evaluate(X_test, y_test)

print("\n", "Loss:", round(loss,3),",", "Accuracy:", round(accuracy,3))
```

```
313/313 [==============================] - 7s 21ms/step - loss: 0.0583 - accuracy: 0.8993

 Loss: 0.058 , Accuracy: 0.899
```

Part 2

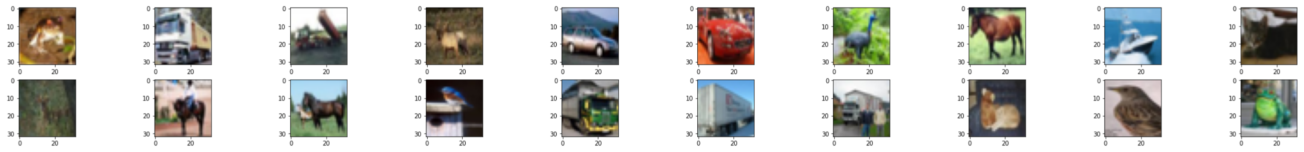
(5pt) Load the dataset using tf.keras.datasets.cifar10.load_data() and show the first 20 images of the training dataset in two rows.

- You will obtain the pair of feature matrix and label vector for the training dataset and the pair of feature matrix and label vector for the testing dataset at the end of this step
- Note that the CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, each with a label value within [0, 9]. In the following step, we want to partition this dataset into two training/testing pairs, one containing images with labels in [0, 4] and the other containing images with labels in [5, 9].

```
#Load the Data
(X_train, y_train), (X_test, y_test)  = cifar10.load_data()

#Train and Test Shape
print('Train: X=%s, y=%s' % (X_train.shape, y_train.shape))
print('Test: X=%s, y=%s' % (X_test.shape, y_test.shape))
```

```
Train: X=(50000, 32, 32, 3), y=(50000, 1)
Test: X=(10000, 32, 32, 3), y=(10000, 1)
```

```
#Display the Training dataset images
plt.figure(figsize=(40, 4))

for i in range(20):
    plt.subplot(2, 20, 2*i+1)
    plt.imshow(X_train[i], cmap=plt.get_cmap('gray'))

plt.show()
```



(5 pt) Reshape the label vectors in both the training and testing datasets to 1D using .reshape(), and compare them with 5 to find out the indices of images that have class labels < 5 and class labels >= 5, respectively, in the training and testing datasets.

- You will obtain four index arrays of Boolean values at the end of this step (<5 and >= 5 for training dataset and <5 and >=5 for testing dataset)
- Hint: label_vector < 5 and label_vector >= 5 will generate such indices

```
print("Original Shape:", y_train.shape, y_test.shape)
print("Original Dimensions:", y_train.ndim,",", y_test.ndim,"\n\n")

#Reshape the label vectors in both the training and testing datasets to 1D using .reshape()
y_train = y_train.reshape(50000)
y_test = y_test.reshape(10000)

print("New Shape:", y_train.shape, y_test.shape)
print("New Dimensions:", y_train.ndim,",", y_test.ndim)
```

```
Original Shape: (50000, 1) (10000, 1)
Original Dimensions: 2 , 2


New Shape: (50000,) (10000,)
New Dimensions: 1 , 1
```

```
#Training
less_index_train = y_train < 5
great_index_train = y_train >= 5

#Testing
less_index_test = y_test < 5
great_index_test = y_test >= 5
```

(5 pt) Use the index arrays obtained in the previous step to split the training/testing dataset into two subsets (each consisting of a feature matrix and a label vector): one with class labels < 5 and one with class labels >= 5. Print out the shapes of the resulting subsets for both training and testing datasets.

- You will obtain four subsets at the end of this step: one pair of training and testing subsets of images with class labels < 5 and another pair of training and testing subsets of images with class labels >= 5.

```
#y_train
y_train_less = y_train[less_index_train]
y_train_greater = y_train[great_index_train]

#y_test
y_test_less = y_test[less_index_test]
y_test_greater = y_test[great_index_test]

#X_train
X_train_less = X_train[y_train_less]
X_train_greater = X_train[y_train_greater]

#X_test
X_test_less = X_test[y_test_less]
X_test_greater = X_test[y_test_greater]


#Group 1: Less Than 5
print("Less Than 5\n",'Train: X=%s, y=%s' % (X_train_less.shape, y_train_less.shape))
print(' Test: X=%s, y=%s' % (X_test_less.shape, y_test_less.shape), "\n\n")

#Group 2: Greater Than 5
print("Greater Than 5\n",'Train: X=%s, y=%s' % (X_train_greater.shape, y_train_greater.shape))
print(' Test: X=%s, y=%s' % (X_test_greater.shape, y_test_greater.shape))
```

```
Less Than 5
```

```
 Train: X=(25000, 32, 32, 3), y=(25000,)
 Test: X=(5000, 32, 32, 3), y=(5000,)


Greater Than 5
 Train: X=(25000, 32, 32, 3), y=(25000,)
 Test: X=(5000, 32, 32, 3), y=(5000,)
```

(5pt) Subtract 5 from the label vectors of the pair of training and testing subsets with class labels >= 5 so that the label vectors in this pair of subsets contains values from 0 to 4. Use to_categorical() to transform all labels into their one-hot encoding forms, and normalize the pixel values of all images into [0, 1].

```
#y_train
y_train_greater = y_train[great_index_train] - 5

#y_test
y_test_greater = y_test[great_index_test] - 5

#To Categorical to transform all labels into one-hot encoded forms
y_train_greater = to_categorical(y_train_greater)
y_test_greater = to_categorical(y_test_greater)

y_train_less = to_categorical(y_train_less)
y_test_less = to_categorical(y_test_less)

#Normalize the pixel values of grayscale images, e.g. rescale them to the range [0,1]
#Convert the data type from unsigned integers to floats
#Divide the pixel values by the maximum value
X_train_greater = X_train_greater.astype('float32') / 255
X_test_greater = X_test_greater.astype('float32') / 255

X_train_less = X_train_less.astype('float32') / 255
X_test_less = X_test_less.astype('float32') / 255
```

(5pt) Build a CNN model_1 using a stack of Conv2D (64 filters of size (3, 3) with ReLU activation), Conv2D (64 filters of size (3, 3) with ReLU activation), MaxPooling2D (pool size of (2, 2)), Dense (128 hidden units with ReLU activation), and output layer. Display the model architecture using .summary().

- You need to specify the correct hyperparameters of the input layer and output layer.

```
#CNN Model Function
def model_defined_2():
  model = models.Sequential()

  #Conv2D (64 filters of size (3, 3) with ReLU activation)
  #Conv2D (64 filters of size (3, 3) with ReLU activation)
  model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)))

  #Conv2D (64 filters of size (3, 3) with ReLU activation)
  model.add(layers.Conv2D(64, (3, 3), activation='relu'))

  #MaxPooling2D (pool size of (2, 2))
  model.add(layers.MaxPooling2D((2, 2)))

  model.add(layers.Flatten())

  #Dense (128 hidden units with ReLU activation)
  model.add(layers.Dense(128, activation='relu'))

  model.add(layers.Dense(5))

  return model
```

```
#Execute Function
cnn_model_2 = model_defined_2()
cnn_model_2.summary()
```

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_38 (Conv2D)          (None, 30, 30, 64)        1792      
                                                                 
 conv2d_39 (Conv2D)          (None, 28, 28, 64)        36928     
                                                                 
 max_pooling2d_19 (MaxPoolin  (None, 14, 14, 64)       0         
 g2D)                                                            
                                                                 
 flatten_19 (Flatten)        (None, 12544)             0         
                                                                 
 dense_41 (Dense)            (None, 128)               1605760   
                                                                 
 dense_42 (Dense)            (None, 5)                 645       
                                                                 
=================================================================
Total params: 1,645,125
Trainable params: 1,645,125
Non-trainable params: 0
_________________________________________________________________
```

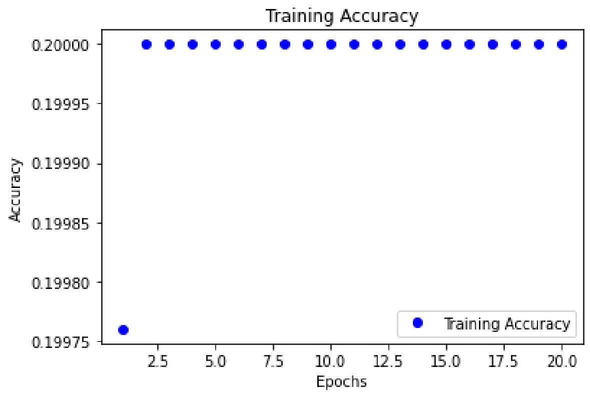
(10pt) Compile and train the model on the subset of training images with class labels < 5 for 20 epochs and batch size of 128. Set verbose = 0 during the training to compress the results. Draw the plot of the training accuracy w.r.t. the epoch number.

```
#Compile Model
cnn_model_2.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

#Train Model with parameters
history = cnn_model_2.fit(X_train_less, y_train_less,
                    epochs=20,
                    batch_size=128)
```

```
Epoch 1/20
196/196 [==============================] - 5s 21ms/step - loss: 4.9018 - accuracy: 0.1998
Epoch 2/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 3/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 4/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 5/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 6/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 7/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 8/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 9/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 10/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 11/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 12/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 13/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 14/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 15/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 16/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 17/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 18/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 19/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
Epoch 20/20
196/196 [==============================] - 4s 20ms/step - loss: 4.9079 - accuracy: 0.2000
```

```
#Create dictionary for loss and accuracy keys (for charting)
history_dict = history.history

#Plotting
plt.clf()
acc = history_dict['accuracy']
loss = history.history['loss']
epochs = range(1, len(acc) + 1)
```

```
plt.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



(5pt) Test your trained model_1 on the subset of testing images with class labels <5 and observe the loss and accuracy using .evaluate().

```
loss, accuracy = cnn_model_2.evaluate(X_test_less, y_test_less)

print("\n", "Loss:", round(loss,3),",", "Accuracy:", round(accuracy,3))
```

```
    157/157 [==============================] - 1s 5ms/step - loss: 4.9079 - accuracy: 0.2000

     Loss: 4.908 , Accuracy: 0.2
```

(10pt) Build a new CNN model_2 that has the same architecture as model_1 and reuse the pre-trained convolutional base layers of model_1 (i.e., all layers before applying flatten()). You need to freeze the pre-trained convolutional base layers of model_2 so that their model parameters will not be changed during the training. Display the model architecture of model_2 using .summary().

- One method to achieve the above step is as follows (You can use other methods as long as they achieve the same goal):

```
#CNN Model Function
def model_defined_3():
  model = models.Sequential()

  #Conv2D (64 filters of size (3, 3) with ReLU activation)
  model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)))

  #Conv2D (64 filters of size (3, 3) with ReLU activation)
  model.add(layers.Conv2D(64, (3, 3), activation='relu'))

  #MaxPooling2D (pool size of (2, 2))
  model.add(layers.MaxPooling2D((2, 2)))

  model.add(layers.Flatten())

  #Dense (128 hidden units with ReLU activation)
  model.add(layers.Dense(128, activation='relu'))

  model.add(layers.Dense(5))

  return model
```

```
#Execute Function
cnn_model_3 = model_defined_3()
cnn_model_3.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_40 (Conv2D)          (None, 30, 30, 64)        1792

 conv2d_41 (Conv2D)          (None, 28, 28, 64)        36928

 max_pooling2d_20 (MaxPoolin  (None, 14, 14, 64)       0
 g2D)

 flatten_20 (Flatten)        (None, 12544)             0

 dense_43 (Dense)            (None, 128)               1605760

 dense_44 (Dense)            (None, 5)                 645

=================================================================
Total params: 1,645,125
Trainable params: 1,645,125
Non-trainable params: 0
_________________________________________________________________
```

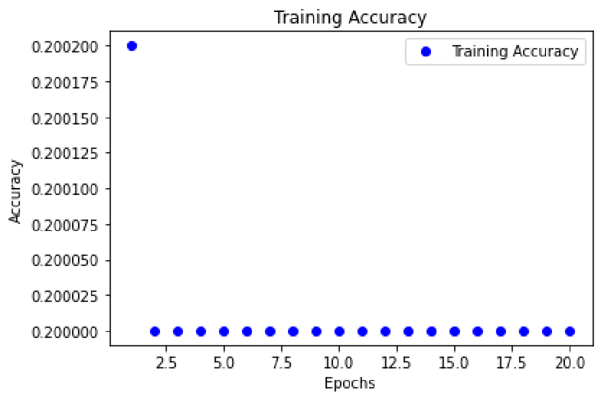
(10pt) Compile model_2, and train it on the subset of training images with class labels >=5 for 20 epochs and batch size of 128. Draw the plot of the training accuracy w.r.t. the epoch number.

```
#Compile Model
cnn_model_3.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
#Train Model with parameters
history = cnn_model_3.fit(X_train_greater, y_train_greater,
                    epochs=20,
                    batch_size=128)
```

```
Epoch 1/20
196/196 [==============================] - 5s 20ms/step - loss: 6.7019 - accuracy: 0.2002
Epoch 2/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 3/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7308 - accuracy: 0.2000
Epoch 4/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 5/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 6/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 7/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7308 - accuracy: 0.2000
Epoch 8/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 9/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 10/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 11/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 12/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 13/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 14/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 15/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 16/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 17/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 18/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7308 - accuracy: 0.2000
Epoch 19/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
Epoch 20/20
196/196 [==============================] - 4s 20ms/step - loss: 6.7307 - accuracy: 0.2000
```

```
#Create dictionary for loss and accuracy keys (for charting)
history_dict = history.history

#Plotting
plt.clf()
acc = history_dict['accuracy']
loss = history.history['loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



(5pt) Test your trained model_2 on the subset of testing images with class labels >=5 and observe the loss and accuracy using .evaluate().

```
loss, accuracy = cnn_model_3.evaluate(X_test_greater, y_test_greater)

print("\n", "Loss:", round(loss,3),",", "Accuracy:", round(accuracy,3))
```

```
    157/157 [==============================] - 1s 6ms/step - loss: 6.7307 - accuracy: 0.2000

     Loss: 6.731 , Accuracy: 0.2
```

✓ 0s completed at 10:15 PM